GP/2121 in cm 02-26-02

Docket No.: GR 98 P 8110 P

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail in an envelope addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231, on the date indicated below.

By: Markus Nolff Date: November 27, 2001

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	:	Christian Siemers
Applic. No.	:	09/816,933
Filed	:	March 23, 2001
Title	:	Program-Controlled Unit
Art Unit	:	2121

CLAIM FOR PRIORITY

Hon. Commissioner of Patents and Trademarks,
Washington, D.C. 20231

Sir:

Claim is hereby made for a right of priority under Title 35, U.S. Code, Section 199, based upon the German Patent Application 198 43 637.8, filed September 23, 1998.

A certified copy of the above-mentioned foreign patent application is being submitted herewith.

Respectfully submitted,

Markus Nolff

MARKUS NOLFF
REG. NO. 37,006

For Applicant

Date: November 27, 2001

Lerner and Greenberg, P.A.
Post Office Box 2480
Hollywood, FL 33022-2480
Tel: (954) 925-1100
Fax: (954) 925-1101

/kf

BUNDESREPUBLIK DEUTSCHLAND

Prioritätsbescheinigung über die Einreichung einer Patentanmeldung

Aktenzeichen:	198 43 637.8
Anmeldetag:	23. September 1998
Anmelder/Inhaber:	Siemens Aktiengesellschaft, München/DE
Bezeichnung:	Programmgesteuerte Einheit
IPC:	G 06 F, G 05 B

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 5. November 2001
Deutsches Patent- und Markenamt
Der Präsident
Im Auftrag

Waasmaier

A 9161
03/00
EDV-L

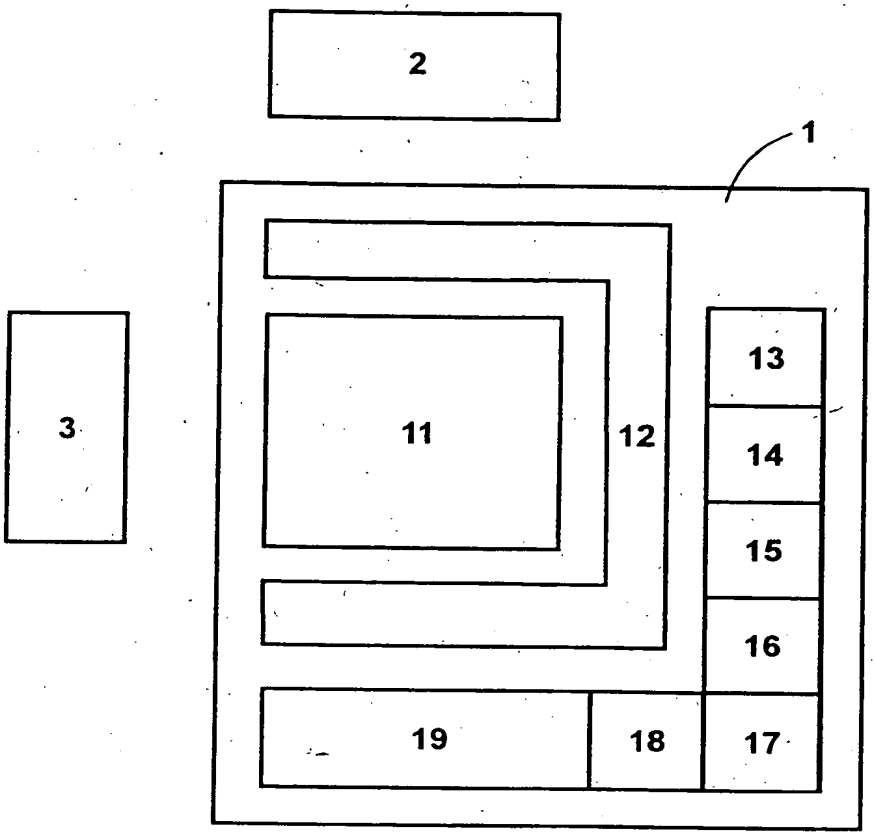


FIG 1

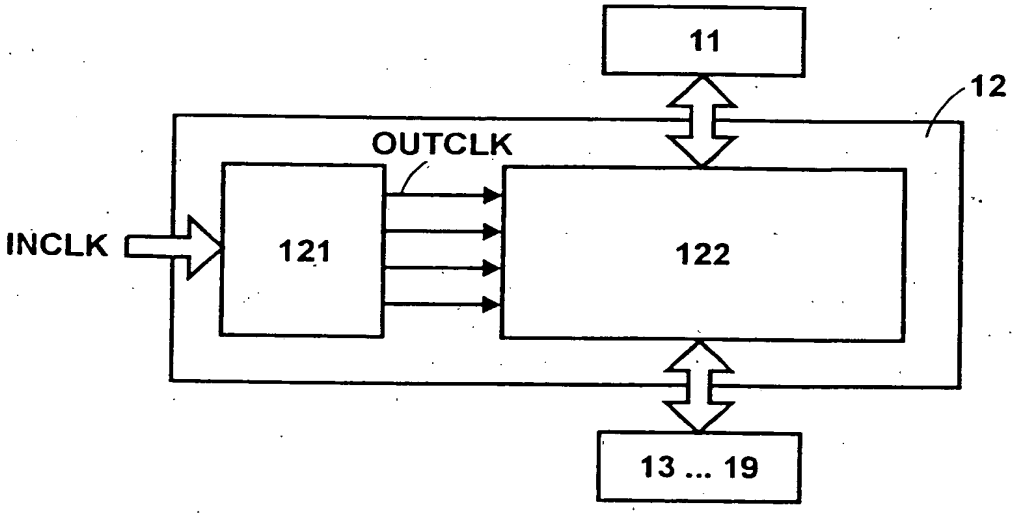


FIG 2

Beschreibung

Programmgesteuerte Einheit

Die vorliegende Erfindung betrifft eine Vorrichtung gemäß dem Oberbegriff des Patentanspruchs 1, d.h. eine programmgesteuerte Einheit.

Programmgesteuerte Einheiten sind durch Software-Programme gesteuerte Einrichtungen, also Mikroprozessoren, Mikrocontroller oder dergleichen; sie existieren in einer Vielzahl verschiedenster Ausführungsformen und bedürfen keiner näheren Beschreibung.

Obgleich Mikroprozessoren, Mikrocontroller und dergleichen in den unterschiedlichsten Ausführungsformen erhältlich sind (oder vielleicht auch gerade deshalb), ist es schwierig oder bisweilen sogar unmöglich, einen Typen zu finden, der die im konkreten Einzelfall gestellten Anforderungen (und möglichst nur diese) erfüllen kann.

Dies trifft in besonderem Maße auf Mikrocontroller zu, denn in diese soll ja nach Möglichkeit die gesamte Peripherie (Analog/Digital-Wandler, Digital/Analog-Wandler, Timer, Interrupt-Controller etc.) integriert sein, die für die betreffende Anwendung benötigt wird.

Wird für die betreffende Anwendung ein Mikrocontroller ausgewählt, welcher - aus welchem Grund auch immer - nicht alle der gestellten Anforderungen erfüllen kann, so wird dadurch in der Regel die Hardware aufwendiger, innerhalb welcher derselbe eingesetzt werden soll. Dies gestaltet sich im Fall von Mikrocontrollern besonders schwierig und aufwendig, weil diese nicht dazu ausgelegt sind, mit Coprozessoren und dergleichen zusammenzuarbeiten.

Wird andererseits für die betreffende Anwendung ein Mikrocontroller ausgewählt, welcher mehr als gefordert leisten kann, so steigt dadurch in der Regel der Preis des denselben enthaltenden Produkts. Hinzu kommt, daß häufig auch der Betrieb der Mikrocontrollers aufwendiger und komplizierter wird. Es müssen nämlich auch die nicht benötigten Peripherie-Einheiten des Mikrocontrollers bei der Systementwicklung berücksichtigt und von dem die auszuführenden Befehle abarbeitenden intelligenten Kern des Mikrocontrollers (dem sogenannten Mikroprozessor-Core bzw. µP-Core) bestimmungsgemäß bedient werden.

Der vorliegenden Erfindung liegt daher die Aufgabe zugrunde, die programmgesteuerte Einheit gemäß dem Oberbegriff des Patentanspruchs 1 derart weiterzubilden, daß diese für eine große Anzahl unterschiedlichster Anwendungsfälle jeweils optimal einsetzbar ist.

Diese Aufgabe wird erfindungsgemäß durch die im kennzeichnenden Teil des Patentanspruchs 1 beanspruchten Merkmale gelöst.

Demnach ist vorgesehen, daß die programmgesteuerte Einheit eine oder mehrere applikationsspezifisch konfigurierbare intelligente Schnittstellen umfaßt.

Bei entsprechender Konfiguration können solche Schnittstellen weitestgehend selbständig dafür sorgen, daß der intelligente Kern und eine oder mehrere Peripherie-Einheiten und/oder der intelligente Kern und eine oder mehrere Speichereinrichtungen und/oder zwei oder mehrere Peripherie-Einheiten untereinander und/oder eine oder mehrere Peripherie-Einheiten und eine oder mehrere Speichereinrichtungen unter minimaler Belastung des intelligenten Kerns wunschgemäß kooperieren.

Gestaltet man die Schnittstellen so, daß diese während des ganz normalen Betriebes der programmgesteuerten Einheit

umkonfigurierbar sind, so kann die programmgesteuerte Einheit sogar dynamisch umkonfiguriert werden.

Es wurde mithin eine programmgesteuerte Einheit geschaffen, welche für eine große Anzahl unterschiedlichster Anwendungsfälle jeweils optimal einsetzbar ist.

Vorteilhafte Weiterbildungen der Erfindung sind den Unteransprüchen, der folgenden Beschreibung und den Figuren entnehmbar.

Die Erfindung wird nachfolgend anhand eines Ausführungsbeispiels unter Bezugnahme auf die Zeichnung näher erläutert. Es zeigen

Figur 1 eine schematische Darstellung des Aufbaus der nachfolgend beschriebenen programmgesteuerten Einheit,

Figur 2 eine schematische Darstellung des Aufbaus einer SLE-Schicht 12 der programmgesteuerten Einheit gemäß Figur 1,

Figur 3 eine schematische Darstellung des Aufbaus einer Taktgenerierungs-Einheit 121 der SLE-Schicht 12 gemäß Figur 2,

Figur 4 eine schematische Darstellung des Aufbaus einer Logikblock-Einheit 122 der SLE-Schicht 12 gemäß Figur 2, und

Figur 5 eine Darstellung zur Veranschaulichung der Konfigurierung der Logikblock-Einheit 122 gemäß Figur 4 für ein praktisches Beispiel.

Die nachfolgend näher beschriebene programmgesteuerte Einheit ist ein Mikrocontroller. Es sei bereits an dieser Stelle darauf hingewiesen, daß es sich bei der programmgesteuerten Ein-

heit grundsätzlich auch um eine beliebige andere programmgesteuerte Einheit wie beispielsweise einen Mikroprozessor handeln kann.

Der betrachtete Mikrocontroller ist in den Figuren mit dem Bezugszeichen 1 bezeichnet. Er zeichnet sich dadurch aus, daß er eine oder mehrere applikationsspezifisch konfigurierbare intelligente Schnittstellen umfaßt.

Dieser neuartige Mikrocontroller-Aufbau wird im folgenden als applikationsspezifisch strukturierbare Controller-Architektur bzw. ASSC-Architektur bezeichnet.

Die generelle Struktur eines die ASSC-Architektur aufweisenden Mikrocontrollers ist schematisch in Figur 1 dargestellt.

Der in der Figur 1 gezeigte Mikrocontroller 1 umfaßt einen meistens als Mikroprozessor-Core bzw. µP-Core bezeichneten, die auszuführenden Befehle abarbeitenden intelligenten Kern 11, eine sogenannte SLE-Schicht 12 sowie Peripherie-Einheiten 13 bis 19, wobei die Peripherie-Einheiten ihrerseits eine serielle Schnittstellen-Einheit 13, eine parallele Schnittstellen-Einheit 14, einen Analog/Digital-Wandler 15, einen Digital/Analog-Wandler 16, einen Timer 17, einen Interrupt-Controller 18 und gegebenenfalls weitere Peripherie-Einheiten 19 umfassen. Ferner sind ein externer Schreib/Lese-Speicher (RAM) 2 und ein externer Festspeicher (ROM) 3 vorgesehen; das RAM 2 und das ROM 3 können problemlos auch als interne Speicher realisiert werden.

Die SLE-Schicht 12 ist schaltungsmäßig zwischen dem µP-Core 11, innerhalb und/oder außerhalb der programmgesteuerten Einheit vorgesehenen Peripherie-Einheiten (beispielsweise den Peripherie-Einheiten 13 bis 19) und/oder Speichereinrichtungen (beispielsweise dem RAM2 und/oder dem ROM 3) angeordnet. Sie enthält strukturierbare Datenpfade und/oder Logikelemente, die sich so strukturieren bzw. konfigurieren

lassen, daß sich die SLE-Schicht 12 als die mindestens eine applikationsspezifisch konfigurierbare Schnittstelle, genauer gesagt als konfigurierbare intelligente Schnittstelle zwischen dem µP-Core und einer oder mehreren Peripherie-Einheiten und/oder zwischen dem µP-Core und einer oder mehreren Speichereinrichtungen und/oder zwischen zwei oder mehreren Peripherie-Einheiten untereinander und/oder zwischen einer oder mehreren Peripherie-Einheiten und einer oder mehreren Speichereinrichtungen verwenden läßt.

Innerhalb der SLE-Schicht 12 existieren vorzugsweise sowohl direkte Verbindungen als auch konfigurierbare Datenpfade und Datenpfadverknüfungen zwischen den über die SLE-Schicht verbundenen oder verbindbaren Einrichtungen; durch die direkten Verbindungen wird erreicht, daß der neuartige Mikrocontroller auch wie ein herkömmlicher (keine SLE-Schicht 12 aufweisender) Mikrocontroller genutzt werden kann.

Wenn die SLE-Schicht 12 auf Speichereinrichtungen (beispielsweise zum RAM 2 und/oder zum ROM 3) Zugriff hat, kann sie für sich und/oder den µP-Core und/oder die Peripherie-Einheiten selbständig, d.h. ohne Mitwirken des µP-Core Daten von und zu den Speichereinrichtungen transferieren. Dadurch lassen sich Datentransfers schneller und ohne Beanspruchung des µP-Core 11 durchführen. Die SLE-Schicht 12 ist in diesem Fall so ausgebildet, daß sie konkurrierende Zugriffe auf Speichereinrichtungen erkennen und handhaben kann.

Die SLE-Schicht 12 kann auch ohne eine Schnittstelle zu den vorhandenen Speichereinrichtungen aufgebaut sein. Dann sind die SLE-Schicht und der µP-Core vorzugsweise so ausgebildet, daß die SLE-Schicht in den µP-Core (genauer gesagt in die Befehls-Pipeline desselben) Befehle injizieren kann, durch welche der µP-Core veranlaßbar ist, von der SLE-Schicht und/oder den Peripherie-Einheiten benötigte Datentransfers durchzuführen.

Der grundsätzliche Aufbau der SLE-Schicht 12 ist in Figur 2 veranschaulicht; sie besteht aus einer Taktgenerierungs-Einheit 121 und einer Logikblock-Einheit 122.

Die Taktgenerierungs-Einheit 121 dient der Versorgung des Logikblock-Einheit 122 mit einem oder mehreren verschiedenen Takten OUTCLK, welche basierend auf einem oder mehreren Master-Takten INCLK generiert werden. In der Taktgenerierungs-Einheit 121 werden beispielsweise Takte mit gegenüber dem oder den Master-Takten INCLK veränderter Frequenz, verändertem Tastverhältnis und/oder veränderter Phasenlage generiert.

Die Taktgenerierungs-Einheit 121 besteht im betrachteten Beispiel aus einer schnellen konfigurierbaren Logik, beispielsweise einer zur praktischen Realisierung einer disjunktiven Normalform geeigneten, sogenannten DNF-Logik (NICHT-UND-ODER-Logik) mit nachgeschalteter konfigurierbarer Speicherung, Invertierung/Pufferung und Rückkopplung. Im betrachteten Beispiel kommt eine zweistufige (zwei DNF-Blöcke aufweisende) DNF-Architektur zum Einsatz.

Der prinzipielle Aufbau einer Taktgenerierungs-Einheit 121, durch welche basierend auf einem Master-Takt INCLK (beispielsweise dem CPU-Takt) ein Ausgangs-Takt OUTCLK generiert wird, ist in Figur 3 veranschaulicht. Es wurde die bei PALs (programmable array logic) und GALs (Generic array logic) übliche Darstellungsform gewählt; transparent dargestellte Knotenpunkte repräsentieren konfigurierbare Verbindungen, gefüllt (schwarz) dargestellte Knotenpunkte repräsentieren feste Verbindungen.

Die Taktgenerierungs-Einheit 121 umfaßt im betrachteten Beispiel zwei identisch aufgebaute DNF-Blöcke DNF1 und DNF2, wobei der erste DNF-Block DNF1 eine erste DNF-Logik-Stufe repräsentiert und aus UND-Gliedern A1, einem ODER-Glied O1, einem (durch den Master-Takt getakteten) Flip-Flop FF1, und

einem Multiplexer MUX1 besteht, und wobei der zweite DNF-Block DNF2 eine zweite DNF-Logik-Stufe repräsentiert und aus UND-Gliedern A2, einem ODER-Glied O2, einem (durch den Master-Takt getakteten) Flip-Flop FF2, und einem Multiplexer MUX2 besteht. Die beiden DNF-Blöcke DNF1 und DNF2 werden mit den selben Eingangssignalen versorgt. Die Eingangssignale umfassen im betrachteten Beispiel den Master-Takt INCLK, den invertierten Master-Takt /INCLK, ein Ausgangssignal OUT1 des ersten DNF-Blocks DNF1, das inverse Ausgangssignal /OUT1, ein Ausgangssignal OUT2 des zweiten DNF-Blocks DNF2, und das inverse Ausgangssignal /OUT2. Das Ausgangssignal OUT2 des zweiten DNF-Blocks DNF2 und das inverse Ausgangssignal /OUT2 werden einem Multiplexer MUX3 zugeführt, und dessen Ausgangssignal wird über einen Treiber T2 als der zu erzeugende Ausgangs-Takt OUTCLK ausgegeben.

Die an die jeweiligen DNF-Blöcke DNF1 und DNF2 angelegten Eingangssignale, genauer gesagt über konfigurierbare Verbindungen auswählbare Signalkombinationen werden an die UND-Glieder A1 bzw. A2 der jeweiligen DNF-Blöcke angelegt. Die Ausgangssignale der UND-Glieder A1 bzw. A2 werden den ODER-Gliedern O1 bzw. O2 zugeführt bzw. als ein Rücksetzsignal ASR zum asynchronen Rücksetzen der Flip-Flops FF1 bzw. FF2 verwendet. Die Ausgangssignale der ODER-Glieder O1 bzw. O2 werden als Eingangssignale der Flip-Flops FF1 bzw. FF2 und verwendet; sie (die Ausgangssignale der ODER-Glieder O1 bzw. O2) und die inversen Ausgangssignale werden zusätzlich an die Multiplexer MUX1 bzw. MUX2 angelegt. Die Ausgangssignale und die inversen Ausgangssignale der Flip-Flops FF1 bzw. FF2 werden ebenfalls als Eingangssignale an die Multiplexer MUX1 bzw. MUX2 angelegt. Das Ausgangssignal und das inverse Ausgangssignal der Multiplexer MUX1 bzw. MUX2 sind die vorstehend bereits erwähnten Ausgangssignale OUT1, /OUT1, OUT2 und /OUT2 der zwei DNF-Blöcke der Taktgenerierungs-Einheit 121.

Ein derartiger oder ähnlicher Aufbau der Taktgenerierungs-Einheit 121 gestattet die Erzeugung einer Vielzahl von Takt-

variationen; durch die Rückkopplung von innerhalb der Taktgenerierungs-Einheit 121 erzeugten (internen) Signalen (im betrachteten Beispiel der Signale OUT1, /OUT1, OUT2, /OUT2) können auch nicht-binäre Tast- und Teilungsverhältnisse realisiert werden.

Die in der Figur 3 gezeigte Anordnung ist nur ein einfaches Beispiel zur Erläuterung des prinzipiellen Aufbaus der Taktgenerierungs-Einheit 121. In der Praxis sollte die Anzahl der durch die Taktgenerierungs-Einheit 121 erzeugten Takte nicht kleiner als vier sein. Die untere Grenze für die mögliche Anzahl der Terme pro DNF sollte ebenfalls vier sein. Pro generiertem Takt sollte mindestens ein internes Signal erzeugt und wie beispielsweise die internen Signale OUT1, /OUT1, OUT2 und /OUT2 als mögliche Basis für die Taktsignalerzeugung bereitgestellt werden.

Die Taktgenerierungs-Einheit 121 muß nicht zwangsläufig unter Verwendung einer zweistufigen DNF-Logik aufgebaut werden. Die verwendete DNF-Logik kann auch weniger Stufen (also nur eine Stufe) oder beliebig viel mehr Stufen aufweisen.

Anstelle der DNF-Logik kann auch eine andere strukturierbare Logik verwendet werden, beispielsweise NAND-Arrays oder multiplexer-basierte Varianten.

Die vorstehend bereits erwähnte Logikblock-Einheit 122 ist der eigentliche Kern der SLE-Schicht 12. Sie kann wie die Taktgenerierungs-Einheit 121 aus einer für die praktische Realisierung der disjunktiven Normalform geeigneten strukturierbaren Logik oder einer der genannten Alternativen bestehen. Durch solche universell einsetzbare strukturierbare Logiken können nahezu beliebige Verknüpfungen, Verarbeitungen und Auswertungen der Eingangssignale durchgeführt werden. Eine derartige Logikblock-Einheit 122 ist äußerst flexibel.

Im vorliegend betrachteten Beispiel wurde die Logikblock-Einheit 122 anders realisiert: sie enthält einen oder mehrere Logikblöcke, die zumindest teilweise in Sub-Blöcke mit vorbestimmten Aufgaben unterteilt sind. Dies vereinfacht die Komplexität der verwendeten Gatter, verringert aber - jedenfalls bei geeigneter Definition der Sub-Blöcke und entsprechenden Querverbindungen zwischen den einzelnen Sub-Blöcken - die Flexibilität der SLE-Schicht 12 kaum.

Der prinzipielle Aufbau eines in verschiedenartige Sub-Blöcke unterteilten Logikblocks ist in Figur 4 veranschaulicht. Der dort gezeigte Logikblock umfaßt vier Sub-Blöcke, nämlich

- einen ersten Sub-Block 5 für die Ein- und/oder Ausgabe von Daten und/oder Signalen und die Verarbeitung derselben,

- einen zweiten Sub-Block 6 für die zentrale Ablaufsteuerung für die innerhalb des betreffenden Logikblocks ablaufenden Vorgänge,

- einen dritten Sub-Block 7 für Adreßberechnungen, und

- einen vierten Sub-Block 8 für die Befehlsinjektionen in den µP-Core.

Die Sub-Blöcke bestehen im betrachteten Beispiel aus konfigurierbaren Multiplexern (zur wunschgemäßen Schaltung von Datenpfaden), Registern (zur Zwischenspeicherung von Daten und/oder codierten Zuständen) und strukturierbarer Logik (zur Verknüpfung von Daten und/oder Signalen miteinander und mit Konstanten und zur Codierung und Decodierung von Zuständen).

Der erste Sub-Block 5 dient im betrachteten Beispiel dazu, eingegebene Daten und/oder Signale untereinander oder mit Konstanten zu verknüpfen; er besteht im betrachteten Beispiel aus einem ersten Multiplexer 51, einem zweiten Multiplexer 52, einem Konstantenregister 53, einer ersten strukturier-

baren Logik 54, einer zweiten strukturierbaren Logik 55, und einem Register 56.

Vom ersten Multiplexer 51 durchgeschaltete Daten und/oder Signale werden untereinander oder mit im Konstantenregister 53 gespeicherten Konstanten durch die strukturierbaren Logiken 54 und 55 verknüpft und über den zweiten Multiplexer 52 ausgegeben. Das zwischen den strukturierbaren Logiken 54 und 55 vorgesehene Register 56 dient als Zwischenspeicher, der beispielsweise dazu verwendbar ist, zu verknüpfende Daten so lange zwischenzuspeichern, bis die anderen Daten, mit denen sie zu verknüpfen sind, gültig sind; die von der strukturierbaren Logik 54 ausgegebenen Daten und/oder Signale können aber auch über einen Bypass direkt (ohne Umweg über das Register 56) zur strukturierbaren Logik 55 übertragen werden.

Die Multiplexer 51 (Quellenauswahl) und 52 (Zielauswahl) verbinden den ersten Sub-Block 5 mit verschiedenen Daten- und/oder Signalquellen und Daten- und/oder Signalzielen. Daten- und/oder Signalquellen und Daten- und/oder Signalziele sind insbesondere die anderen Sub-Blöcke, der µP-Core 11, innerhalb und/oder außerhalb des Mikrocontrollers vorgesehene Peripherie-Einheiten (z.B. die Peripherie-Einheiten 13 bis 19) und/oder Speicher (z.B. das RAM 2 und/oder das ROM 3). Die Verbindung zu den genannten oder anderen Daten- und/oder Signalquellen und Daten- und/oder Signalzielen kann (muß aber nicht) über Datenbusse erfolgen.

Der erste Sub-Block 5 ist wirkungsmäßig mit der arithmetisch-logischen Einheit (ALU) einer nach dem Von-Neumann-Prinzip arbeitenden programmgesteuerten Einheit vergleichbar.

Der zweite Sub-Block 6 ist im betrachteten Beispiel eine Statemachine zur zentralen Ablaufsteuerung im betreffenden Logikblock; er besteht im betrachteten Beispiel aus einer ersten strukturierbaren Logik 61, einer zweiten strukturierbaren Logik 62, einem Register 63, und einem Multiplexer 64.

Der strukturierbaren Logik 61 obliegt es, insbesondere in Abhängigkeit von den Zuständen, Signalen und Signalverläufen in bzw. von der Taktgenerierungs-Einheit 121, den anderen Sub-Blöcken, dem µP-Core 11, den Peripherie-Einheiten (z.B. den Peripherie-Einheiten 13 bis 19) und/oder den Speichern (z.B. dem RAM 2 und/oder dem ROM 3) den jeweiligen Zustand des betreffenden Logikblocks zu ermitteln; vom jeweiligen Zustand des Logikblocks hängt es ab, welcher Vorgang in diesem gerade ablaufen soll (muß). Der besagte Zustand, genauer gesagt ein diesen repräsentierender Datenwert kann im Register 63 gespeichert und bei Bedarf durch die strukturierbare Logik 62 decodiert werden. Die strukturierbare Logik 62 gibt ein oder mehrere Ausgangssignale aus. Diese Ausgangssignale sind auf dem jeweiligen Zustand des betreffenden Logikblocks basierende Steuersignale und werden über den Multiplexer 64 insbesondere an die anderen Sub-Blöcke, gegebenenfalls aber auch an den µP-Core, die Peripherie-Einheiten und/oder die Speichereinrichtungen ausgeben.

Durch den zweiten Sub-Block 6 lassen sich im betrachteten Beispiel verschiedene einfache Automatentypen (bis zu Mealy) realisieren, und zwar auch mit verschiedenen Takten.

Die dem zweitem Sub-Block 6 obliegende Aufgabe ist mit dem Steuerwerk einer Von-Neumann-CPU vergleichbar, wobei die Bearbeitung von Befehlen naturgemäß entfällt (die Zustände werden von äußeren Signalen getriggert durchlaufen).

Der dritte Sub-Block 7 dient im betrachteten Beispiel insbesondere (aber nicht ausschließlich) der Adreßberechnung für blockweise Datentransfers; er besteht im betrachteten Beispiel aus einem Adreßregister 71, einem Konstantenregister 72, einer Inkrementier/Dekrementier-Einheit 73, einer ersten strukturierbaren Logik 74, und einer zweiten strukturierbaren Logik 75.

Der dritte Sub-Block 7 kann Quellen- und Zieladressen errechnen und diese an die anderen Sub-Blöcke, den µP-Core, die Peripherie-Einheiten und/oder den Speicher ausgeben.

Die jeweils aktuelle Adresse ist im Adreßregister 71 gespeichert und kann durch die Inkrementier/Dekrementier-Einheit 73 inkrementiert oder dekrementiert werden. Das Konstantenregister 72 ist dazu ausgelegt, eine Endadresse zu speichern, und durch die strukturierbaren Logiken kann ein Adreßvergleich durchgeführt werden (um beispielsweise überprüfen zu können, ob die im Adreßregister 71 gespeicherte Adresse die im Konstantenregister 72 gespeicherte Endadresse erreicht hat).

Das Vorsehen dieses dritten Sub-Blocks 7 ist optional und ist insbesondere (aber nicht ausschließlich) dann sehr nützlich, wenn die SLE-Schicht 12 einen direkten Zugriff zum Speicher hat.

Der vierte Sub-Block 8 dient im betrachteten Beispiel zur Injektion von Befehlen in die Pipeline des µP-Core 11; er besteht im betrachteten Beispiel aus einem Konstantenregister 81 und einer strukturierbaren Logik 82.

Die Injektion von Befehlen erfolgt unter Steuerung durch den zweiten Sub-Block 6, genauer gesagt durch die von diesem ausgegebenen Steuersignale. Der Befehlscode des zu injizierenden Befehls ist im Konstantenregister 81 gespeichert und wird bei Bedarf (gegebenenfalls zusammen mit Adreß- und Dateninformationen) über die strukturierbare Logik 82 an den µP-Core 11 übermittelt; die strukturierbare Logik 82 wird hierzu bei Bedarf vom zweiten Sub-Block 6 in einen Zustand versetzt, in dem der im Konstantenregister 81 gespeicherte Befehlscode (gegebenenfalls zusammen mit den aus dem ersten Sub-Block 5 ausgegebenen Daten und/oder der aus dem dritten Sub-Block 7 ausgegebenen Adresse) in die Pipeline des µP-Core injiziert wird.

Wie vorstehend zumindest teilweise schon erwähnt oder angedeutet wurde, bestehen zwischen den einzelnen Sub-Blöcken 5 bis 8 diverse Querverbindungen. Die in der Figur 4 eingezeichneten Querverbindungen sind nur als beispielhaft anzusehen. Die Anzahl der Querverbindungen sowie deren Anfangs- und Endpunkte hängen insbesondere von der konkreten Anwendung des betreffenden Logikblocks und der Anzahl und der Funktion der Sub-Blöcke desselben ab und können bei Bedarf als konfigurierbare Verbindungen ausgelegt sein.

Die zumindest teilweise Aufteilung der Logikblöcke der Logikblock-Einheit 122 der SLE-Schicht 12 in Sub-Blöcke ermöglicht die praktische Realisierung der Logikblöcke mit minimalem Aufwand. Insbesondere müssen erheblich weniger Verknüpfungsmöglichkeiten vorgesehen werden als bei der Realisierung der Logikblock-Einheit durch eine oder mehrere universell einsetzbare strukturierbare Logiken; dadurch, daß jeder Sub-Block nur eine genau definierte Aufgabe erfüllen muß, müssen die Sub-Blöcke nämlich - anders als universell einsetzbare strukturierbare Logiken - jeweils nur einige wenige Signale miteinander verknüpfen können.

Die Aufgaben, die die jeweiligen Sub-Blöcke zu erfüllen haben, und die Querverbindungen zwischen den einzelnen Sub-Blöcken ermöglichen es, daß die betreffenden Logikblöcke ähnlich leistungsfähig sind wie unter Verwendung von universell einsetzbaren strukturierbaren Logiken aufgebaute Logikblöcke.

Dies gilt auch dann, wenn die Logikblöcke in Sub-Blöcke aufgeteilt sind, die anders aufgebaut sind und/oder andere Aufgaben zu erfüllen haben und/oder andere Querverbindungen aufweisen und/oder in einer größeren oder kleineren Anzahl vorgesehen sind als die im vorliegend betrachteten Beispiel zum Einsatz kommenden und unter Bezugnahme auf die Figur 4 beschriebenen Sub-Blöcke 5 bis 8.

Bei Datengruppenoperationen sind gegebenenfalls zusätzlich Speichereinheiten in die SLE-Schicht 12 zu integrieren. Dann müssen für spätere Berechnungen benötigte Ergebnisse nicht in außerhalb der SLE-Schicht 12 vorgesehene Speicher ausgelagert werden, sondern können gleich intern innerhalb der SLE-Schicht 12 (zwischen-)gespeichert werden.

Die strukturierbaren Logiken, die in den Sub-Blöcken zum Einsatz kommen, sind im betrachteten Beispiel DNF-Logiken, deren Aufbau im wesentlichen dem Aufbau der in der Taktgenerierungs-Einheit 121 verwendeten DNF-Logiken entspricht; die Ein- und Ausgangssignale sind dann natürlich keine Taktsignale, sondern Daten und/oder Steuersignale, und benötigte Taktsignale (beispielsweise zur Taktung der Flip-Flops) werden aus den von der Taktgenerierungs-Einheit 121 generierten Taktsignalen ausgewählt.

Es besteht allerdings keine Einschränkung darauf, daß die strukturierbaren Logiken der Logikblock-Einheit 122 solche oder ähnliche DNF-Logiken sind; es können statt dessen auch NAND-Arrays, multiplexer-basierte Varianten oder sonstige strukturierbare Logiken zum Einsatz kommen.

Die Konfigurierung der konfigurierbaren Elemente der SLE-Schicht, d.h. die Konfigurierung der Multiplexer, der konfigurierbaren Verbindungen innerhalb der strukturierbaren Logiken und der Register kann dem Wesen nach wie die Konfigurierung der bekannten feldprogrammierbaren Logiken (PLAs, GALs, PLDs, FPGAs etc.) erfolgen.

Eine erste Möglichkeit hierzu besteht im (irreversiblen) Herstellen bzw. Löschen von Verbindungen unter Verwendung sogenannter Fuses oder Antifuses.

Eine andere Möglichkeit besteht in der Durchführung einer reversiblen Konfigurierung basierend auf die gewünschte Konfiguration repräsentierenden Daten, wobei die Daten in inner-

halb oder außerhalb der programmgesteuerten Einheit vorgesehenen EPROMs, EEPROMs oder dergleichen gespeichert werden. Dadurch kann die Konfiguration der programmgesteuerten Einheit eine begrenzte Anzahl von Malen geändert werden.

Eine weitere Möglichkeit besteht in der Durchführung einer reversiblen Konfigurierung basierend auf die gewünschte Konfiguration repräsentierenden Daten, wobei die Daten jedoch in einem RAM oder dergleichen gespeichert werden. Dadurch kann die Konfiguration der programmgesteuerten Einheit unbegrenzt oft und sehr schnell geändert werden.

Die EPROMs, EEPROMs, RAMs oder sonstigen Speicher, in denen die die Konfiguration der programmgesteuerten Einheit repräsentierenden Daten gespeichert sind, sind vorzugsweise so ausgebildet, daß sie in den Speicher- oder I/O-Bereich einblendbar sind, auf den der µP-Core 11 zugreifen kann. Dann kann die Konfiguration der SLE-Schicht 12 durch den µP-Core 11 selbst wunschgemäß eingestellt werden.

Wenn ausgeschlossen werden soll, daß durch die Konfiguration der SLE-Schicht 12 der Betrieb der programmgesteuerten Einheit gestört wird, kann vorgesehen werden, die Konfiguration nur zu vorbestimmten Zeitpunkten (beispielsweise innerhalb einer vorbestimmten Zeit nach dem Rücksetzen der programmgesteuerten Einheit) zu gestatten.

Bei entsprechender Konfigurierung der SLE-Schicht 12 können dieser bestimmte Aufgaben übertragen werden, deren Abarbeitung bislang ausschließlich oder weitestgehend durch den µP-Core 11 erfolgen konnte oder mußte. Zu diesen Aufgaben gehören insbesondere (aber nicht ausschließlich) die Vorverarbeitung, Nachverarbeitung, Auswertung und/oder Kontrolle von Daten und/oder Signalen sowie die Veranlassung und Überwachung der Kooperation der über die SLE-Schicht 12 verbundenen Einrichtungen.

Läßt man die Konfiguration oder eine Umkonfiguration der SLE-Schicht 12 jederzeit zu, so können diese Aufgaben sogar dynamisch in die SLE-Schicht ausgelagert werden.

Unabhängig davon, also selbst dann, wenn keine dynamische Konfigurierung der SLE-Schicht 12 möglich ist, kann der µP-Core 11 durch die SLE-Schicht 12 beträchtlich entlastet werden. Dies wird nachfolgend anhand eines praktischen Beispiels erläutert.

Das Beispiel betrifft eine Analog/Digital-Wandlung von Daten. Genauer gesagt soll

- ein A/D-Wandler (beispielsweise der A/D-Wandler 15) mit einer Wandlungsbreite von 8 Bit von einem Timer (beispielsweise vom Timer 17) gestartet werden,

- das Ergebnis der A/D-Wandlung zusammen mit einer 12-Bit-Zählmarke gespeichert werden,

- das Ergebnis der A/D-Wandlung auf das Über- und Unterschreiten bestimmter Grenzwerte überwacht werden, wobei bei einem Über- oder Unterschreiten der Grenzwerte in eine bestimmte Routine zu verzweigen ist,

- und der Vorgang nach 2048 Messungen abgebrochen werden.

Eine derartige Anwendung erfordert bei einer reinen Softwarelösung einen relativ hohen Aufwand. Da ein typischer A/D-Wandler, der in einem Mikrocontroller integriert ist, nicht spontan das Ergebnis liefert, also als sogenannter Flash-Wandler arbeitet und eine Wandlungszeit im Bereich einiger Mikrosekunden besitzt, muß bei exakter Ausführung der Anwendungsspezifikation einer der folgenden Wege beschritten werden:

1) Der Timer löst einen Interrupt aus. Die Interrupt-Serviceroutine startet die A/D-Wandlung und wird dann beendet. Der A/D-Wandler löst bei Beendigung der Wandlung ebenfalls einen Interrupt aus. Durch die daraufhin ausgeführte Interrupt-Serviceroutine wird das A/D-Wandlungsergebnis ausgelesen und verarbeitet.

2) Der Timer löst einen Interrupt aus. In der daraufhin ausgeführten Interrupt-Serviceroutine wird die A/D-Wandlung gestartet, auf das Wandlungsende gewartet, und schließlich (nach dem Wandlungsende) das A/D-Wandlungsergebnis ausgelesen und verarbeitet.

Wenn die Wandlungszeiten kürzer als die Interrupt-Latenzzeiten sind, ist der zweiten Variante der Vorzug zu geben. Ansonsten wäre die erste Variante zu bevorzugen. Am günstigsten ist jedoch im allgemeinen die folgende dritte Variante:

3) Der Timer löst einen Interrupt aus. In der daraufhin ausgeführten Interrupt-Serviceroutine wird das letzte A/D-Wandlungsergebnis ausgelesen, die nächste A/D-Wandlung gestartet, und das ausgelesene A/D-Wandlungsergebnis ausgewertet.

Bei dieser dritten Variante erfolgt das Lesen und Auswerten des A/D-Wandlungsergebnis zwar in der Regel später als es eigentlich möglich wäre. Diese Verzögerung ist jedoch im allgemeinen tolerierbar.

Der Aufwand, der zur praktischen Realisierung der dritten Variante zu treiben ist, ist zweifellos am geringsten. Nichtsdestotrotz dauert die Abarbeitung der durch den Timer-Interrupt zur Ausführung gebrachten Interrupt-Serviceroutine mindestens (d.h. wenn ein Befehl pro Takt ausgeführt wird und wenn keine Über- oder Unterschreitungen der Grenzwerte durch die A/D-Wandlungsergebnisse auftreten und wenn keine Interrupt-Latenzzeiten vorhanden wären) 30 Takte.

Verlagert man die Ausführung dieser Anwendung so weit wie möglich in die SLE-Schicht 12, so läßt sie sich erheblich schneller durchführen.

Der in der Figur 4 gezeigte und unter Bezugnahme darauf beschriebene Logikblock ist dann beispielsweise so zu konfigurieren, daß die in Figur 5 gezeigte Struktur entsteht.

Die Eingangssignale der in der Figur 5 gezeigten Anordnung sind die mit ad[0..7] bezeichneten A/D-Wandlungsergebnisse des A/D-Wandlers 15 sowie die mit Start_T bezeichneten Impulse des Timers 17.

Die strukturierbaren Elemente des Logikblocks gemäß Figur 5, werden so konfiguriert, daß dieser bei einem Über- oder Unterschreiten der Grenzwerte durch die A/D-Wandlungsergebnisse und bei Erreichen des Endes der Meßreihe mit Irq-Con bezeichnete Interrupt Requests absetzt und zum Speichern der A/D-Wandlungsergebnisse zusammen mit der Zählmarke entsprechende Befehle in die Befehls-Pipeline des µP-Core injiziert. Dabei obliegt

- dem zweiten Sub-Block 6 die zentrale Ablaufsteuerung innerhalb des betreffenden Logikblocks,

- dem ersten Sub-Block 5 der Vergleich der vom A/D-Wandler 15 erhaltenen A/D-Wandlungsdaten ad[0..7] mit im Konstantenregister 53 gespeicherten Werten u[0..7] (oberer Grenzwert) und l[0..7] (unterer Grenzwert) und die Erzeugung der Interrupt Requests Irq_Con,

- dem dritten Sub-Block 7 der Vergleich von Adressen (zur Vervollständigung der zu injizierenden Befehle und zur Ermittlung des Endes der Meßreihe), und

- dem vierten Sub-Block 8 die Generierung bzw. Zusammenstellung und die Injizierung der zu injizierenden Befehle.

Die strukturierbaren Logiken 61 und 62 sind dabei so konfiguriert, daß sie basierend auf Eingangssignalen MClk, Start_T, End_Of_Adr und Reset unter Verwendung von internen Signalen S0, S1, S2 und End_Reached wie durch die folgenden Boolschen Gleichungen definiert Ausgangssignale Irq_Gen, Inject_1, Inject_2, New_Adr und Read erzeugt:

In der strukturierbaren Logik 61:

```
S0 = /S2 * /S0 * Start_T * /End_Reached  +
     /S2 * S1 * /S0 * /End_Reached;
S1 = /S2 * /S1 * S0 * /End_Reached  +
     /S2 * S1 * /S0 * /End_Reached;
S2 = /S2 * S1 * S0;
S0.CLK = MClk;                          /* Taktung mit MClk */
S1.CLK = MClk;                          /* Taktung mit MClk */
S2.CLK = MClk;                          /* Taktung mit MClk */
/* Dies beschreibt den Durchlauf durch 5 Zustände,
getriggert durch Start_T */

End_Reached = End_Of_Adr + End_Reached * /Reset;
End_Reached.CLK = MClk;                 /* Taktung mit MClk */
/* Gespeichertes Ende-Flag */
```

In der strukturierbaren Logik 62:

```
New_Adr = /S2 * S1 * /S0 + /S2 * S1 * S0;
/* Signal für Zustand 2 und 3 */

Inject_1 = /S2 * S1 * /SO;
/* Injektion 1. Move-Befehl */

Inject_2 = /S2 * S1 * SO;
```

```
/* Injektion 2. Move-Befehl */

Read = /S2 * /S1 * SO;
/* Auslösen des Lesevorgangs */

Irq_Gen = End_Reached;
/* Auslösung Interrupt Request bei Ende */
```

Die strukturierbaren Logiken 54 und 55 sind so konfiguriert, daß sie basierend auf Eingangssignalen ad[0..7], u[0..7], l[0..7], Read, Irq_Gen unter Verwendung von internen Signalen tmp_upper[0..3], tmp_lower[0..3], tmp_eq[0..3] (für Zwischenergebnisse), tmp_ad[0..7] (für Zwischenspeicherung nach Abfrage) und Irq_Tmp1 (für IRQ-Generierung und Rücksetzen) wie durch die folgenden Boolschen Gleichungen definiert Ausgangssignale Irq_Con1, Irq_Con2, b_ad[0..7] erzeugt:

In der strukturierbaren Logik 54:

```
tmp_upper[z] = /u[2z+1] *  ad[2z+1]  +
                u[2z+1] *  ad[2z+1] * /u[2z] * ad[2z]  +
               /u[2z+1] * /ad[2z+1] * /u[2z] * ad[2z];
tmp_lower[z] =  l[2z+1] * /ad[2z+1]  +
                l[2z+1] *  ad[2z+1] * l[2z] * /ad[2z]
               /l[2z+1] * /ad[2z+1] * l[2z] * /ad[2z];
tmp_eq[z] =  u[2z+1] *  ad[2z+1] *  u[2z] *  ad[2z]..+
             u[2z+1] *  ad[2z+1] * /u[2z] * /ad[2z]  +
            /u[2z+1] * /ad[2z+1] *  u[2z] *  ad[2z]  +
            /u[2z+1] * /ad[2z+1] * /u[2z] * /ad[2z];
/* z läuft von 0 bis 3, Vergleich auf größer/kleiner und
Gleichheit */

tmp_upper[z].CLK = /Read;  /* Taktung bei Ausleseende */
tmp_lower[z].CLK = /Read;  /* Taktung bei Ausleseende */
tmp_eq[z].CLK = /Read;     /* Taktung bei Ausleseende */
```

```
tmp_ad[x] = ad[x]            /* x von 0 .. 7 */

tmp_ad[x].CLK = /Read ;      /* Taktung bei Ausleseende */

Irq_Tmp1 = 1;
Irq_Tmp1.CLK = Irq_Gen;          /* Taktung mit Irq_Gen*/
Irq_Tmp1.AR = Irq1Reset;  /* Rücksetzen mit Irq1Reset */
/* Generierung IRQ für Aufnahmeende */
```

In der strukturierbaren Logik 55:

```
Irq_Con1 = Irq_Tmp1;
/* IRQ wird für Aufnahmeende generiert */

Irq_Con2 = tmp_upper[3] + tmp_eq[3] * tmp_upper[2]  +
           tmp_eq[3] * tmp_eq[2] * tmp_upper[1]  +
           tmp_eq[3] * tmp_eq[2] * tmp_eq[1] *
                                          tmp_upper[0]  +
           /* Overflow */
           tmp_lower[3] + tmp_eq[3] * tmp_lower[2]  +
           tmp_eq[3] * tmp_eq[2] * tmp_lower[1]  +
           tmp_eq[3] * tmp_eq[2] * tmp_eq[1] *
                                          tmp_lower[0];
          /* Underflow */

b_ad[y] = tmp_ad[y]            /* y läuft von 0 .. 7 */
```

Die strukturierbaren Logiken 74 und 75 sind so konfiguriert, daß sie basierend auf Eingangssignalen A[0..15], EA[0..15] unter Verwendung von internen Signalen temp[0..7] wie durch die folgenden Boolschen Gleichungen definiert Ausgangssignale End_Adr und BA[0..15] erzeugt:

In der strukturierbaren Logik 74:

```
temp[x] = /A[2x] * /EA[2x] * /A[2x+1] * /EA[2x+1]  +
           A[2x] *  EA[2x] * /A[2x+1] * /EA[2x+1]  +
          /A[2x] * /EA[2x] *  A[2x+1] *  EA[2x+1]  +
           A[2x] *  EA[2x] *  A[2x+1] *  EA[2x+1];
/* x läuft von 0 bis 7, Vergleich auf Gleichheit */
```

In der strukturierbaren Logik 75:

```
End_Adr = temp[0] * temp[1] * temp[2] * temp[3] *
          temp[4] * temp[5] * temp[6] * temp[7];
/* Ergibt 1, falls Adressen und Vergleichsadresse iden-
tisch */

BA[y] = A[y];                    /* y läuft von 0 .. 15 */
```

Die strukturierbare Logik 82 ist so konfiguriert, daß sie basierend auf Eingangssignalen Inject_1, Inject_2, BA[0..15], b_ad[0..7], und CR4[0..15] wie durch die folgenden Boolschen Gleichungen definiert Ausgangssignale Inject und MP[0..31] (angenommenes 32-Bit-Interface zum µP-Core) erzeugt:

```
Inject = Inject_1 + Inject_2;
MP[0..15] = CR4[0..15] * Inject_1  +
            CR4[0..15] * Inject_2;
/* der in CR4[0..15] gespeicherte */
/* 16-Bit Befehlscode wird übermittelt */

MP[16..23] = BA[0..7] * Inject_1  +
             b_ad[0..7] * Inject_2;
MP[24..31] = BA[8..15] * Inject_1;
/* der oder die Operanden */
/* (16-Bit-Adresse oder 8-Bit-AD-Wert) */
/* werden übermittelt */
```

Wie sich aus den vorstehenden Ausführungen ergibt, durchläuft die Anordnung gemäß Figur 5 bei jedem Auslesen und Verarbeiten von A/D-Wandlungsdaten aus dem A/D-Wandler vier Zustände, nämlich

1) Lesen des A/D-Wandlungsergebnisses ad[0..7] aus dem A/D-Wandler mit automatischem Neustart des A/D-Wandlers.

2) Das A/D-Wandlungsergebnis wird verarbeitet, d.h. das A/D-Wandlungsergebnis wird mit einem im Konstantenspeicher 53 gespeicherten oberen Grenzwert u[0..7] und einem ebenfalls im Konstantenspeicher gespeicherten unteren Grenzwert l[0..7] verglichen. Zugleich wird ein erster Move-Befehl in die Pipeline des µP-Core injiziert.

3) Ein zweiter Move-Befehl wird injiziert. Dieser besteht aus einem Transfer des A/D-Wandlungsergebnisses auf die inzwischen inkrementierte Adresse. Wenn das A/D-Wandlungsergebnis außerhalb des durch die Grenzwerte definierten Bereiches liegt, wird gleichzeitig ein Interrupt Request ausgelöst.

4) Eventuell wird das Ende der 2048 Wandlungen erreicht. In diesem Fall wird ein Flag gesetzt, das ein weiteres Auslesen und Verarbeiten von A/D-Wandlungsergebnissen verhindert. Weiterhin wird zur Signalisierung des Endes der Routine ein weiterer Interrupt Request ausgelöst.

Geht man davon aus, daß jeder Zustand genau einen Ablauftakt benötigt, so werden für jedes Auslesen und Verarbeiten von A/D-Wandlungsdaten aus dem A/D-Wandler 4 Takte benötigt; der µP-Core wird durch die injizierten Befehle zeitlich mit zwei oder drei CPU-Takten belastet.

Das vorliegend betrachtete Lesen und Auswerten von A/D-Wandlungsergebnissen läßt sich bei Miteinbeziehung der SLE-Schicht 12 in einem kleinen Bruchteil der Zeit durchführen, die nötig wäre, wenn man wie bislang vorgehen würde, also die komplette Ablaufsteuerung und Datenverarbeitung im wesentlichen ausschließlich dem µP-Core 11 überlassen würde.

Selbstverständlich lassen sich auch bei gänzlich anderen Anwendungen als dem vorstehend beschriebenen Beispiel ähnliche Vorteile erzielen.

Die beschriebene programmgesteuerte Einheit ist dadurch für eine große Anzahl unterschiedlichster Anwendungsfälle jeweils optimal einsetzbar.

Patentansprüche

1. Programmgesteuerte Einheit,
d a d u r c h g e k e n n z e i c h n e t ,
daß diese eine oder mehrere applikationsspezifisch konfigurierbare intelligente Schnittstellen umfaßt.

2. Programmgesteuerte Einheit nach Anspruch 1,
d a d u r c h g e k e n n z e i c h n e t ,
daß die mindestens eine applikationsspezifisch konfigurierbare intelligente Schnittstelle durch eine strukturierbare Hardware (12) gebildet wird, die schaltungsmäßig zwischen einem auszuführende Befehle abarbeitenden intelligenten Kern (11) der programmgesteuerten Einheit (1), innerhalb oder außerhalb der programmgesteuerten Einheit vorgesehenen Peripherie-Einheiten (13 bis 19) und/oder Speichereinrichtungen (2, 3) angeordnet ist.

3. Programmgesteuerte Einheit nach Anspruch 2,
d a d u r c h g e k e n n z e i c h n e t ,
daß die strukturierbare Hardware (12) als konfigurierbare intelligente Schnittstelle zwischen dem intelligenten Kern (11) und einer oder mehreren Peripherie-Einheiten (13 bis 19) und/oder zwischen dem intelligenten Kern (11) und einer oder mehreren Speichereinrichtungen (2, 3) und/oder zwischen zwei oder mehreren Peripherie-Einheiten (13 bis 19) untereinander und/oder zwischen einer oder mehreren Peripherie-Einheiten (13 bis 19) und einer oder mehreren Speichereinrichtungen (2, 3) konfigurierbar ist.

4. Programmgesteuerte Einheit nach Anspruch 2 oder 3,
d a d u r c h g e k e n n z e i c h n e t ,
daß die strukturierbare Hardware (12) mit einer Vielzahl von möglichen Daten- und/oder Signalquellen und Daten- und/oder Signalzielen verbunden ist, und daß Multiplexer (51, 52, 64) zur Auswahl der aktuellen Daten- und/oder Signalquellen und der aktuellen Daten- und/oder Signalziele vorgesehen sind.

5. Programmgesteuerte Einheit nach Anspruch 4, dadurch gekennzeichnet, daß die Daten- und/oder Signalquellen und die Daten- und/oder Signalziele den intelligenten Kern (11), die Peripherie-Einheiten (13 bis 19), die Speichereinrichtungen (2, 3) und/oder Teile der strukturierbaren Hardware (12) selbst enthaltene Einheiten umfassen.

6. Programmgesteuerte Einheit nach einem der Ansprüche 2 bis 5, dadurch gekennzeichnet, daß die strukturierbare Hardware (12) in der Lage ist, erhaltene Daten und/oder Signale auszuwerten und/oder zu verarbeiten.

7. Programmgesteuerte Einheit nach einem der Ansprüche 2 bis 6, dadurch gekennzeichnet, daß die strukturierbare Hardware (12) in der Lage ist, Befehle in die Befehls-Pipeline des die auszuführenden Befehle abarbeitenden intelligenten Kerns (11) der programmgesteuerten Einheit zu injizieren.

8. Programmgesteuerte Einheit nach einem der Ansprüche 2 bis 7, dadurch gekennzeichnet, daß die strukturierbare Hardware (12) in der Lage ist, Interrupt Requests oder sonstige Ereignisse signalisierende Meldungen zu generieren und auszugeben.

9. Programmgesteuerte Einheit nach einem der Ansprüche 2 bis 8, dadurch gekennzeichnet, daß die strukturierbare Hardware (12) in der Lage ist, auf Interrupt Requests oder sonstige Ereignisse signalisierende

Meldungen von mit ihr verbundenen Einrichtungen zu reagieren und/oder deren Weiterleitung zu unterbinden.

10. Programmgesteuerte Einheit nach einem der Ansprüche 2 bis 9,
dadurch gekennzeichnet,
daß durch die Strukturierung der strukturierbaren Hardware (12) Datenpfade veränderbar und/oder Logikelemente konfigurierbar sind.

11. Programmgesteuerte Einheit nach einem der Ansprüche 2 bis 10,
dadurch gekennzeichnet,
daß die strukturierbare Hardware (12) eine Taktgenerierungs-Einheit (121) und eine Logikblock-Einheit (122) umfaßt, wobei es der Taktgenerierungs-Einheit (121) obliegt, ein oder mehrere Taktsignale (OUTCLK, MClk) für die Logikblock-Einheit (122) zu generieren, und wobei es der Logikblock-Einheit (122) obliegt, über die strukturierbare Hardware (12) zu verbindende Einrichtungen wunschgemäß kooperieren zu lassen.

12. Programmgesteuerte Einheit nach Anspruch 11,
dadurch gekennzeichnet,
daß sowohl die Taktgenerierungs-Einheit (121) als auch die Logikblock-Einheit (122) konfigurierbare Elemente enthalten.

13. Programmgesteuerte Einheit nach Anspruch 11 oder 12,
dadurch gekennzeichnet,
daß die Taktgenerierungs-Einheit (121) zumindest teilweise durch eine DNF-Logik, ein NAND-Array, eine multiplexerbasierte Logikvariante oder eine sonstige strukturierbare Logik gebildet wird.

14. Programmgesteuerte Einheit nach einem der Ansprüche 11 bis 13,
dadurch gekennzeichnet,

daß die Logikblock-Einheit (122) zumindest teilweise durch eine DNF-Logik, ein NAND-Array, eine multiplexer-basierte Logikvariante oder eine sonstige strukturierbare Logik gebildet wird.

15. Programmgesteuerte Einheit nach einem der Ansprüche 11 bis 14,
dadurch gekennzeichnet,
daß die Logikblock-Einheit (122) ein oder mehrere Logikblöcke umfaßt, die zumindest teilweise in individuell konfigurierbare Sub-Blöcke (5 bis 8) mit vorbestimmten Aufgaben unterteilt sind.

16. Programmgesteuerte Einheit nach Anspruch 15,
dadurch gekennzeichnet,
daß einer der Sub-Blöcke (5 bis 8) dazu ausgelegt ist, als Verarbeitungseinrichtung zur arithmetischen und/oder logischen Verarbeitung eingegebener Daten verwendet zu werden.

17. Programmgesteuerte Einheit nach Anspruch 15 oder 16,
dadurch gekennzeichnet,
daß einer der Sub-Blöcke (5 bis 8) dazu ausgelegt ist, als Statemachine zur zentralen Ablaufsteuerung verwendet zu werden.

18. Programmgesteuerte Einheit nach einem der Ansprüche 15 bis 17,
dadurch gekennzeichnet,
daß einer der Sub-Blöcke (5 bis 8) dazu ausgelegt ist, als Adreßberechnungseinrichtung zur Berechnung von Quellen- und Zieladressen verwendet zu werden.

19. Programmgesteuerte Einheit nach einem der Ansprüche 15 bis 18,
dadurch gekennzeichnet,
daß einer der Sub-Blöcke (5 bis 8) dazu ausgelegt ist, als Befehlsinjektionseinrichtung zur Injektion von Befehlen in

die Befehls-Pipeline des intelligenten Kerns (11) der programmgesteuerten Einheit verwendet zu werden.

20. Programmgesteuerte Einheit nach einem der Ansprüche 2 bis 19,
dadurch gekennzeichnet,
daß die strukturierbare Hardware (12) unter Verwendung von Fuses und/oder Anti-Fuses konfigurierbar ist.

21. Programmgesteuerte Einheit nach einem der Ansprüche 2 bis 20,
dadurch gekennzeichnet,
daß die strukturierbare Hardware (12) durch reversible Maßnahmen konfigurierbar ist.

22. Programmgesteuerte Einheit nach Anspruch 21,
dadurch gekennzeichnet,
daß die strukturierbare Hardware (12) basierend auf die gewünschte Konfiguration repräsentierenden Daten konfigurierbar ist, und daß diese Daten in Speichereinrichtungen gespeichert sind, die in den durch den intelligenten Kern (11) der programmgesteuerten Einheit ansprechbaren Speicher- oder I/O-Bereich einblendbar sind.

23. Programmgesteuerte Einheit nach einem der Ansprüche 2 bis 22,
dadurch gekennzeichnet,
daß die Konfigurierung der strukturierbaren Hardware (12) nur zu vorbestimmten Zeiten möglich ist.

24. Programmgesteuerte Einheit nach einem der Ansprüche 1 bis 22,
dadurch gekennzeichnet,
daß die Konfigurierung der strukturierbaren Hardware (12) jederzeit möglich ist.

Zusammenfassung

Programmgesteuerte Einheit

Die beschriebene programmgesteuerte Einheit zeichnet sich dadurch aus, daß sie eine oder mehrere applikationsspezifisch konfigurierbare intelligente Schnittstellen umfaßt. Sie ist dadurch für eine große Anzahl unterschiedlichster Anwendungsfälle jeweils optimal einsetzbar.

Figur 1

Bezugszeichenliste

1	Mikrocontroller
2	RAM
3	ROM
5	erster Sub-Block
6	zweiter Sub-Block
7	dritter Sub-Block
8	vierter Sub-Block
11	µP-Core
12	SLE-Schicht
13	serielle Schnittstellen-Einheit
14	parallele Schnittstellen-Einheit
15	Analog/Digital-Wandler
16	Digital/Analog-Wandler
17	Timer
18	Interrupt-Controller
19	weitere Peripherie-Einheiten
51	erster Multiplexer
52	zweiter Multiplexer
53	Konstantenregister
54	erste strukturierbare Logik
55	zweite strukturierbare Logik
56	Register
61	erste strukturierbare Logik
62	zweite strukturierbare Logik
63	Register
64	Multiplexer
71	Adreßregister
72	Konstantenregister
73	Inkrementier/Dekrementier-Einheit
74	erste strukturierbare Logik

75	zweite strukturierbare Logik
81	Konstantenregister
82	strukturierbare Logik
121	Taktgenerierungs-Einheit
122	Logikblock-Einheit
INCLK	Master-Takt für die Taktgenerierungs-Einheit 121
OUTCLK	Ausgangs-Takt der Taktgenerierungs-Einheit 121
DNF1	erster DNF-Block
DNF2	zweiter DNF-Block
Ax	UND-Glieder
Ox	ODER-Glieder
FFx	Flip-Flops
Tx	Treiber
MUXx	Multiplexer
ASR	Asynchrones Rücksetzsignal
OUT1	Ausgangssignal des ersten DNF-Blocks DNF1
/OUT1	inverses Ausgangssignal des ersten DNF-Blocks DNF1
OUT2	Ausgangssignal des zweiten DNF-Blocks DNF2
/OUT2	inverses Ausgangssignal des zweiten DNF-Blocks DNF2

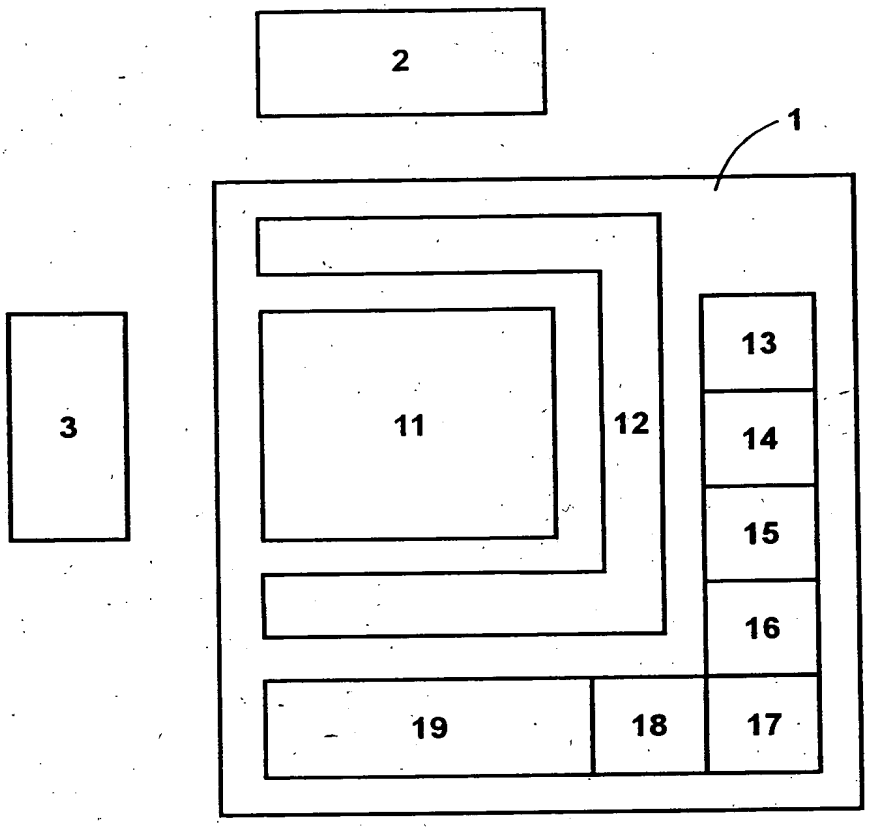


FIG 1

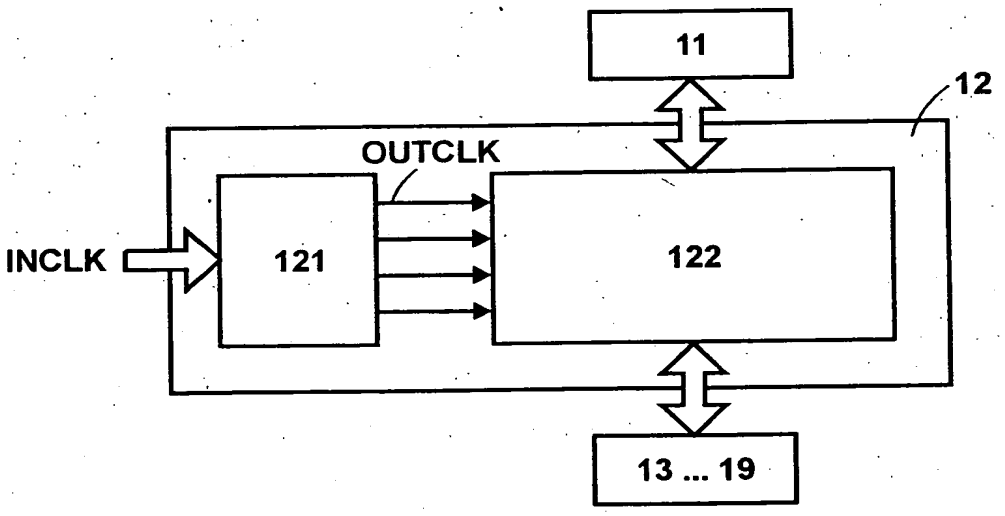


FIG 2

FIG 3

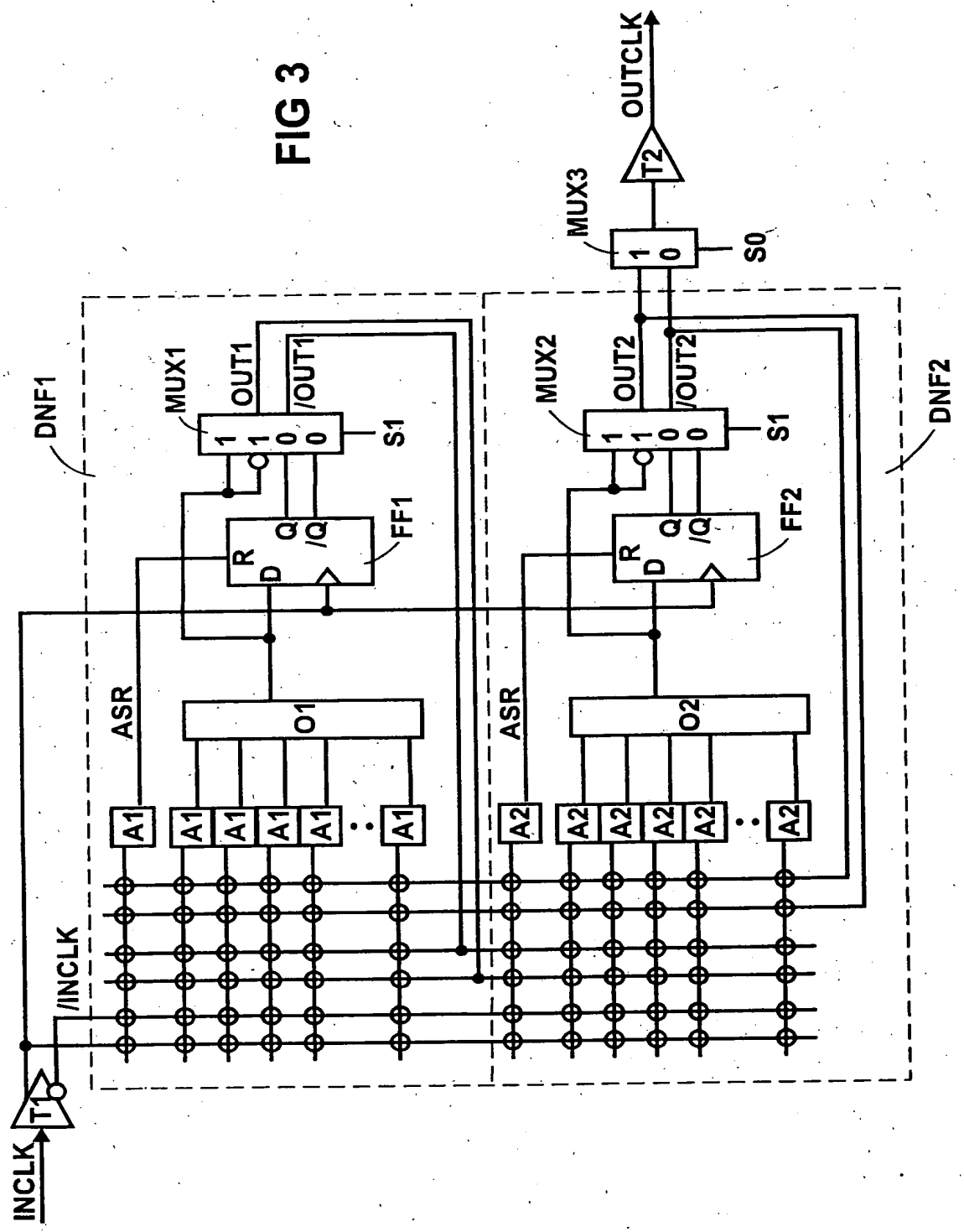

INCLK
/INCLK
T1
T2
OUTCLK
DNF1
DNF2
MUX1
MUX2
MUX3
OUT1
/OUT1
OUT2
/OUT2
S0
S1
FF1
FF2
R
D
Q
/Q
ASR
O1
O2
A1
A2

FIG 4

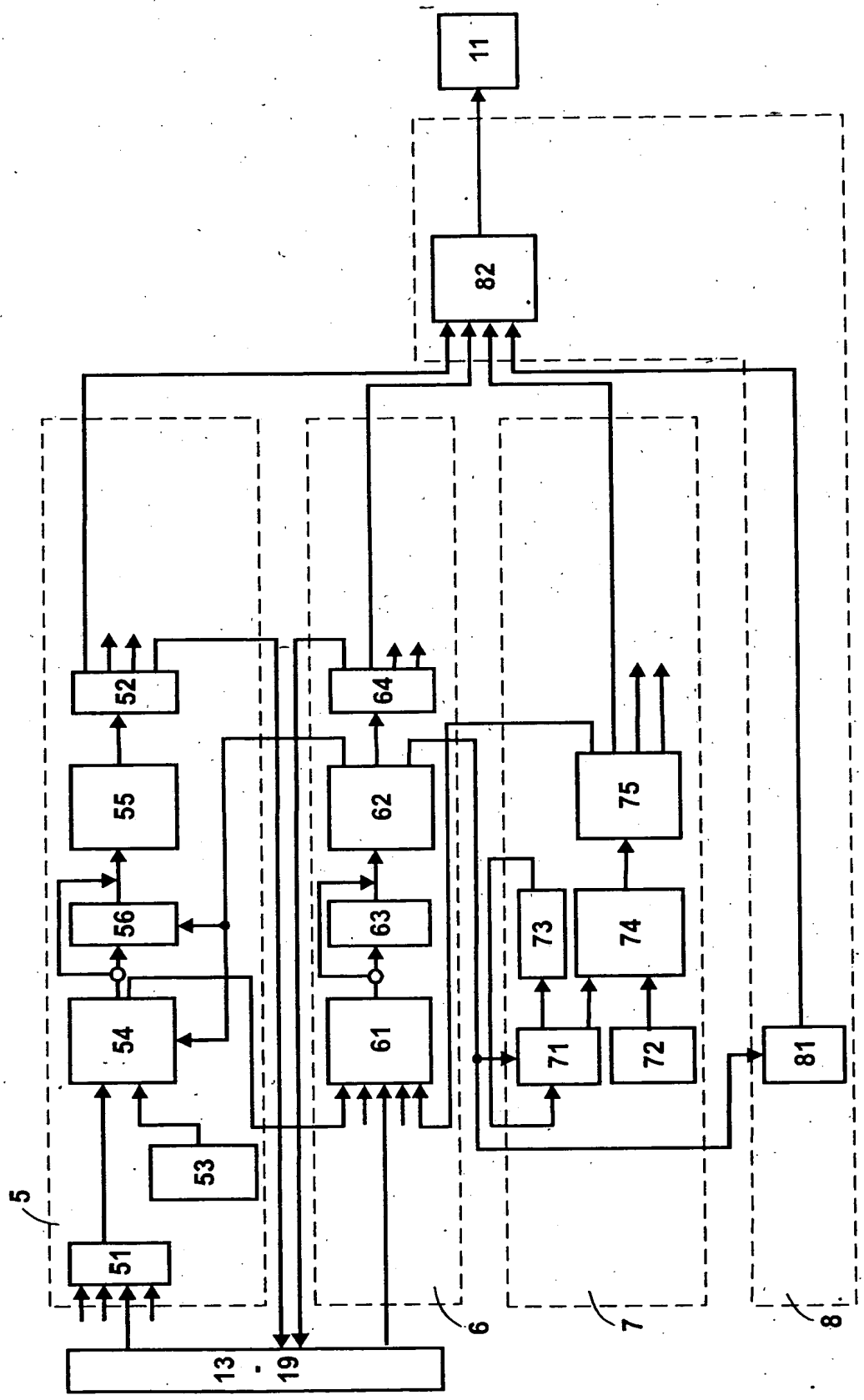
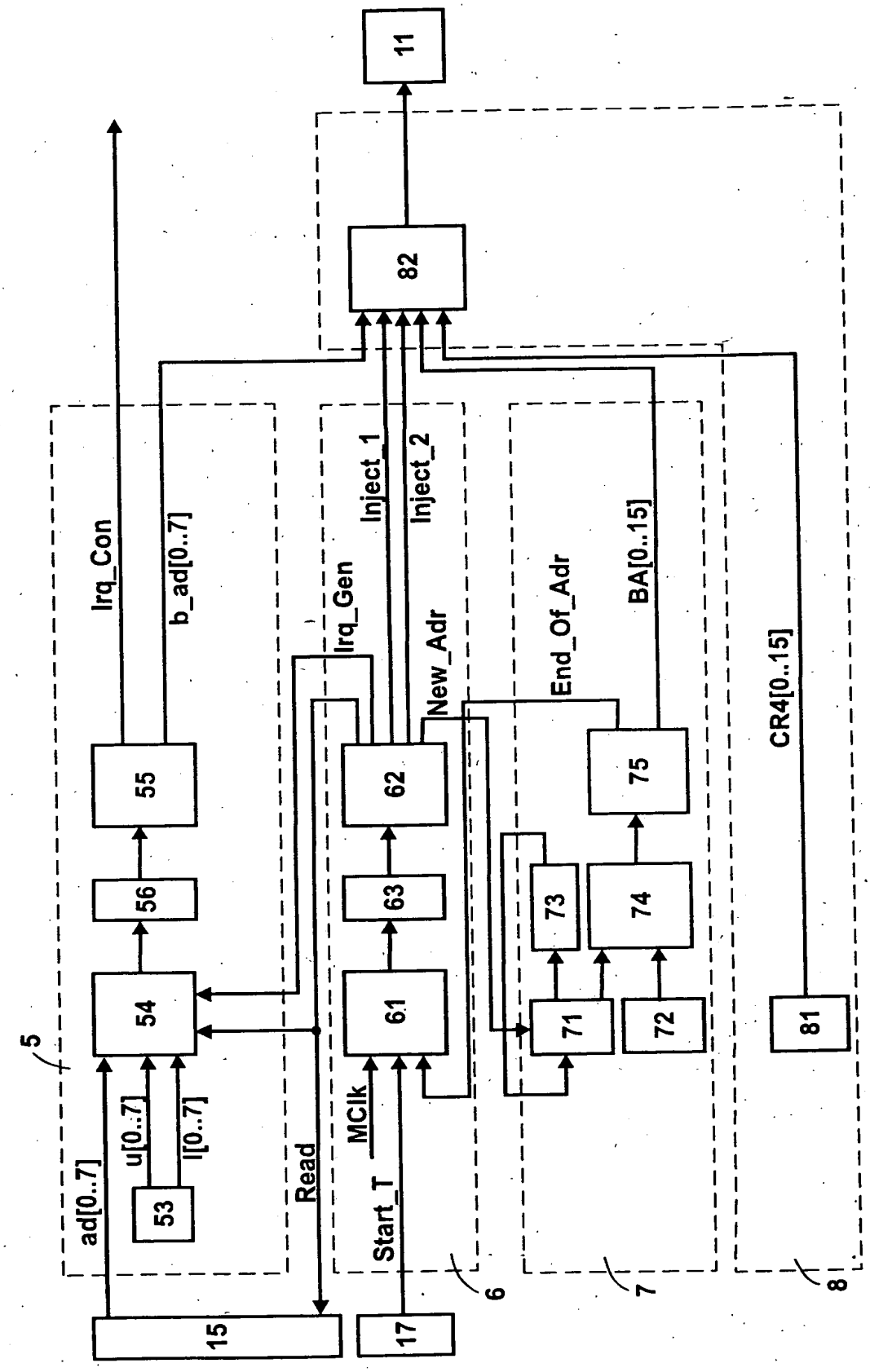


FIG 5


11
82
Irq_Con
b_ad[0..7]
Irq_Gen
Inject_1
Inject_2
New_Adr
End_Of_Adr
BA[0..15]
CR4[0..15]
55
56
54
62
63
61
75
73
74
71
72
81
53
ad[0..7]
u[0..7]
l[0..7]
Read
MClk
Start_T
15
17
5
6
7
8